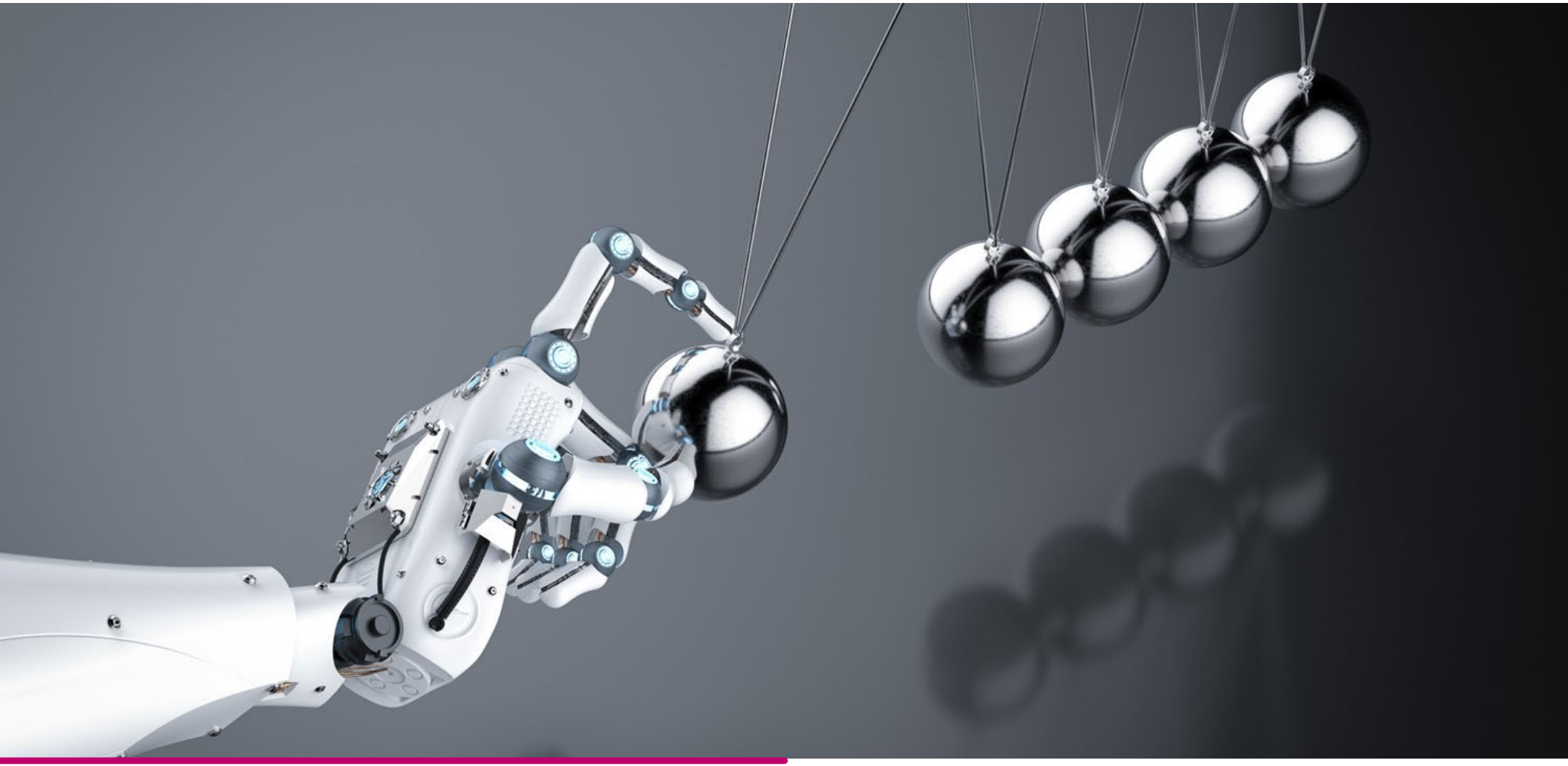
IT-Introduction (BBA)

Business Applications

Prof. Dr. Peter Weber



Wir geben Impulse

COMPETENCE CENTER eCOMMERCE

Fachhochschule Südwestfalen
University of Applied Sciences

Contents

5.1 System Software, Development Software and Application Software

- **Software** as a general term for programs written in programming language

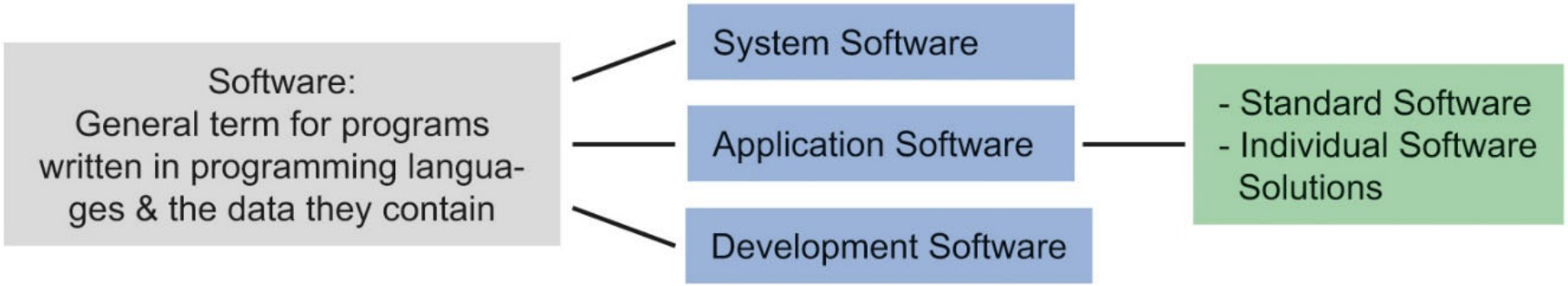
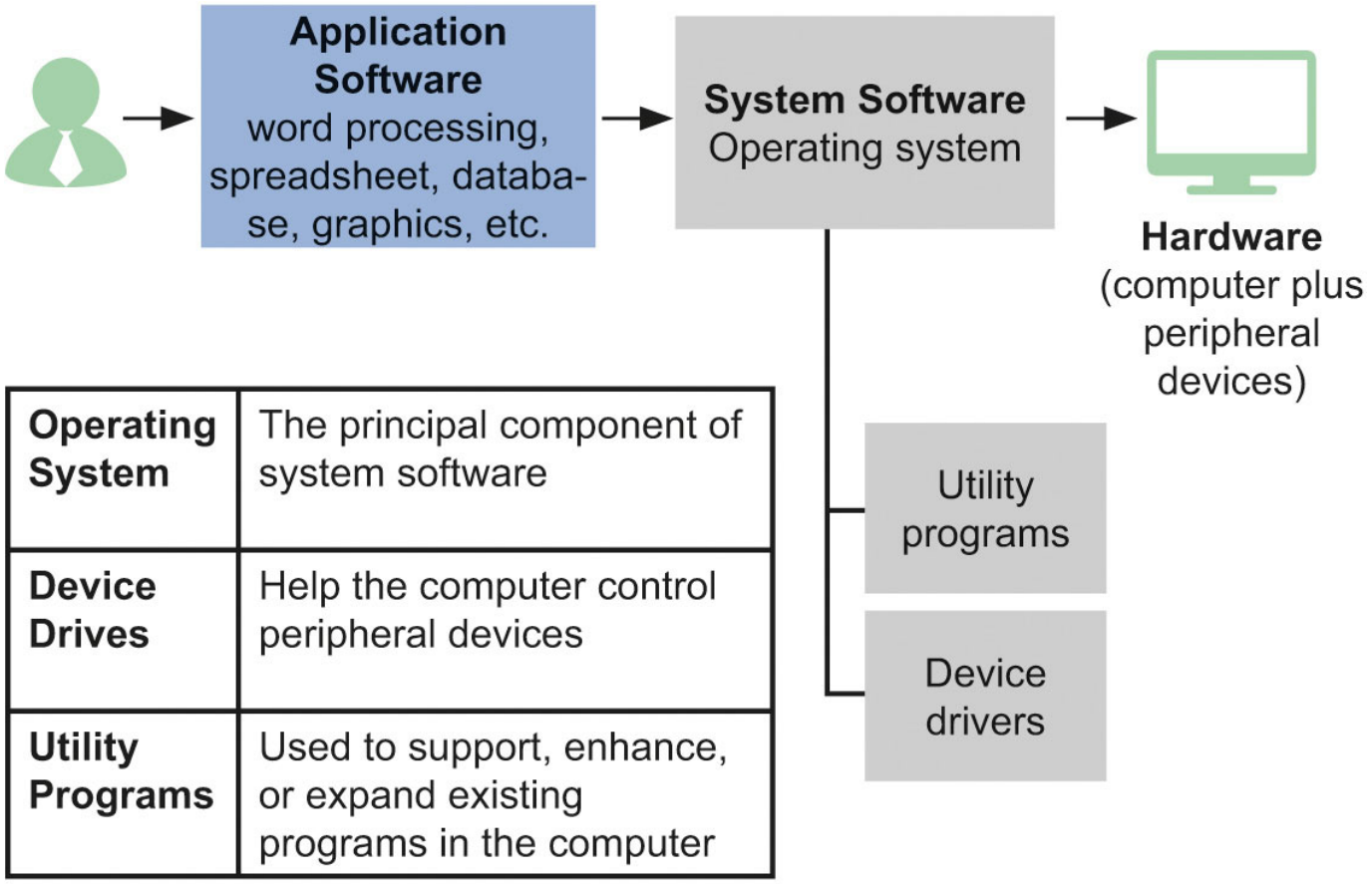


Fig. 5.1: Three Types of Software.

- **Development Software:**
  - Provides tools and methodologies for creating, executing, testing and correcting programs
  - Development software / environments for programming languages

5.1 System Software, Development Software and Application Software



Operating System	The principal component of system software
Device Drives	Help the computer control peripheral devices
Utility Programs	Used to support, enhance, or expand existing programs in the computer

Fig. 5.2: Classification of Systems Software.

5.1 System Software, Development Software …

Excursus: Operating Systems

Tasks and discussion questions:

- Which operating systems (OS) do you know for computers and for mobile devices? What are their differences?
- Do some research and find the current market shares of the three most popular computer operating systems.
- Which core functions does an operating system have?
- Discuss the myth of macOS and Linux being invulnerable and threat-proof. What do you think?

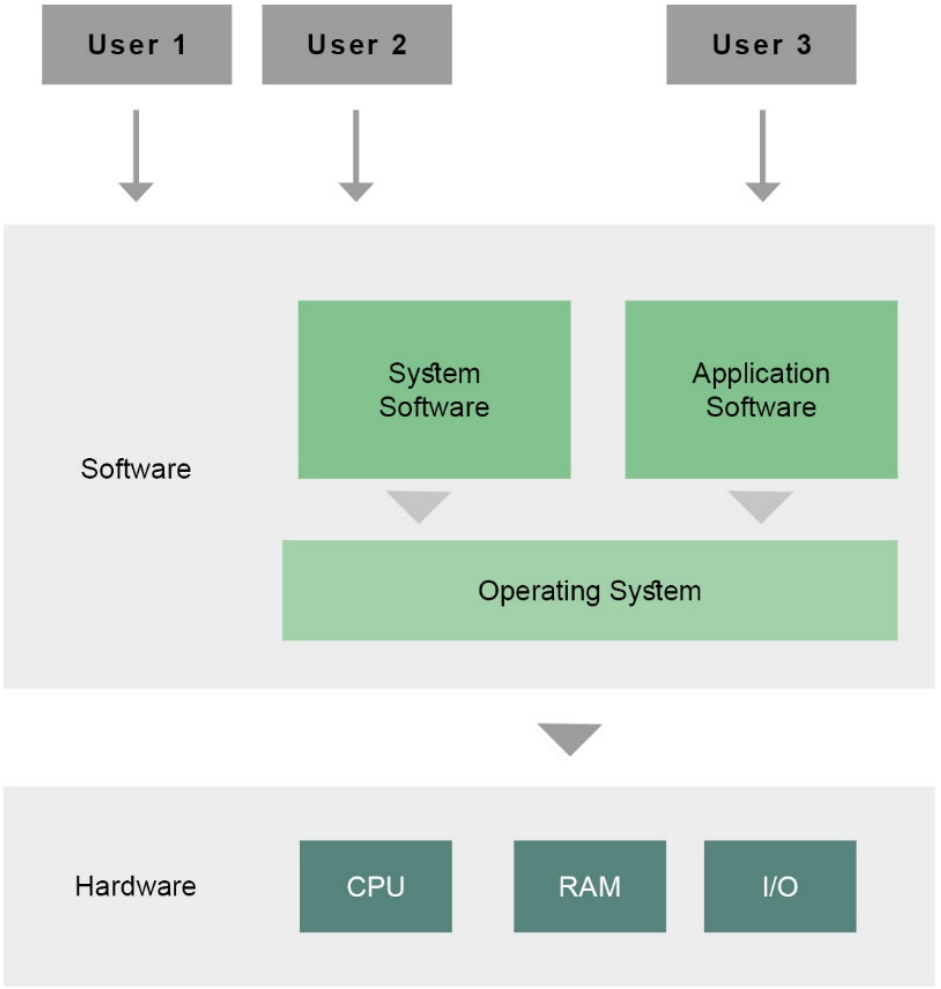


Fig. 5.3 User, Software, Operating System and Hardware.

5.1 System Software, Development Software and Application Software

- **Supports users** in completing their tasks and in solving specific problems
- **Different Types:**
  - **Standard Software**
    - Prefabricated programs that can be used by different companies or organizations for similar tasks
    - e.g. email software, antivirus software, office software
  - **Individual Software Solutions**
    - Designed as solutions for specific company requirements
    - Normally associated with much higher costs than standard software
  - **Hybrids**
    - Standard software products that are adapted to the company needs via customizing and reconfiguration (e.g. ERP systems)

5.2 Application Systems

Classification

Figure 5.4: Organization Levels of a Company

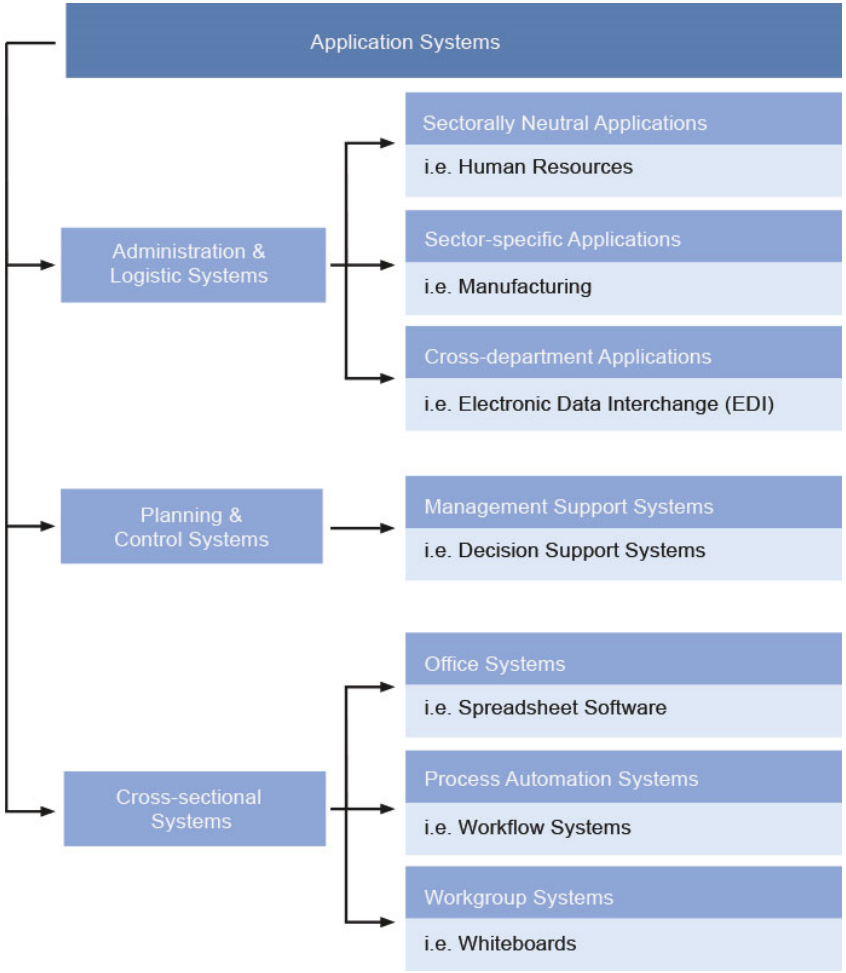


Fig. 5.5: Classification of Application Systems.

5.3 Integrated Information Systems

The Information Systems Pyramid

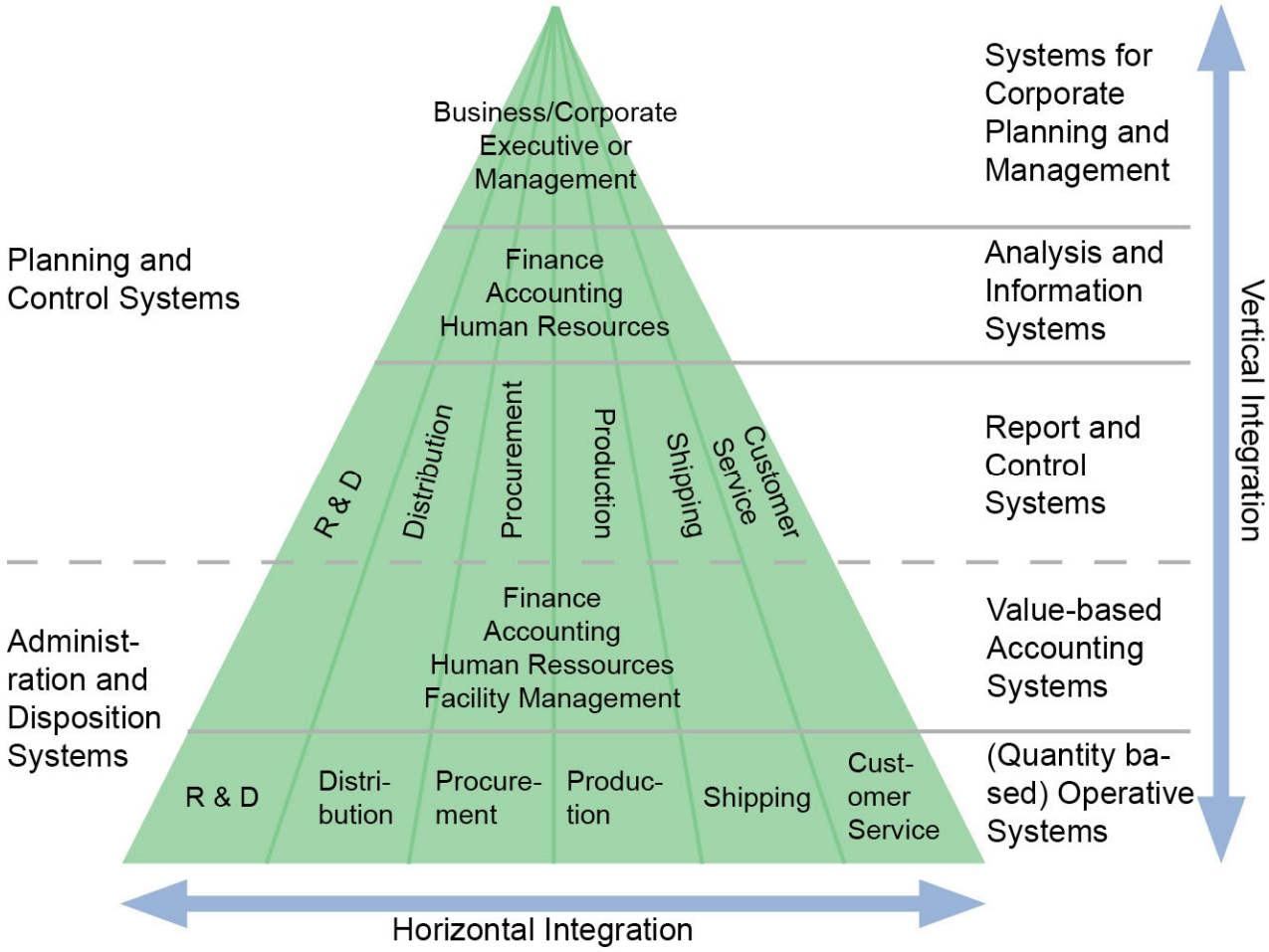


Fig. 5.8: The Information Systems Pyramid [based on [Mert13, p. 19]].

5.3 Integrated Information Systems

Dimensions of Integration

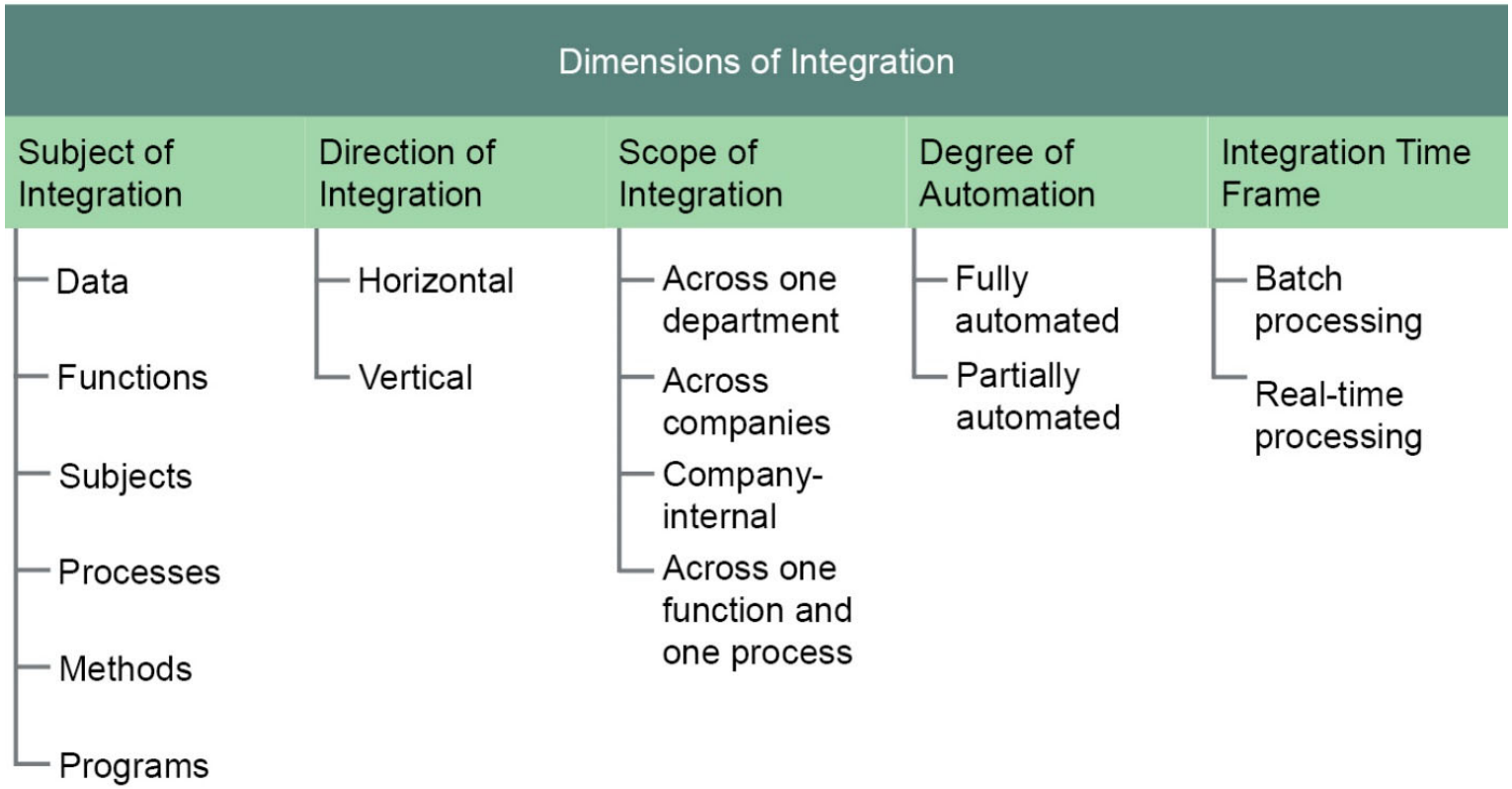


Fig. 5.9: Dimensions of Integration [LLS10, p.466] (transl.).

5.3 Integrated Information Systems

(Non-)Integrated Application Systems

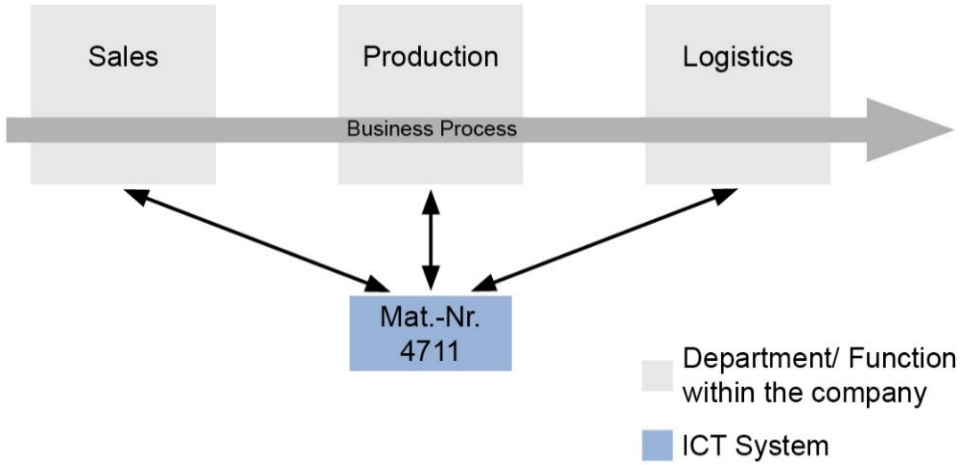


Fig. 5.11: Integrated Application Systems.

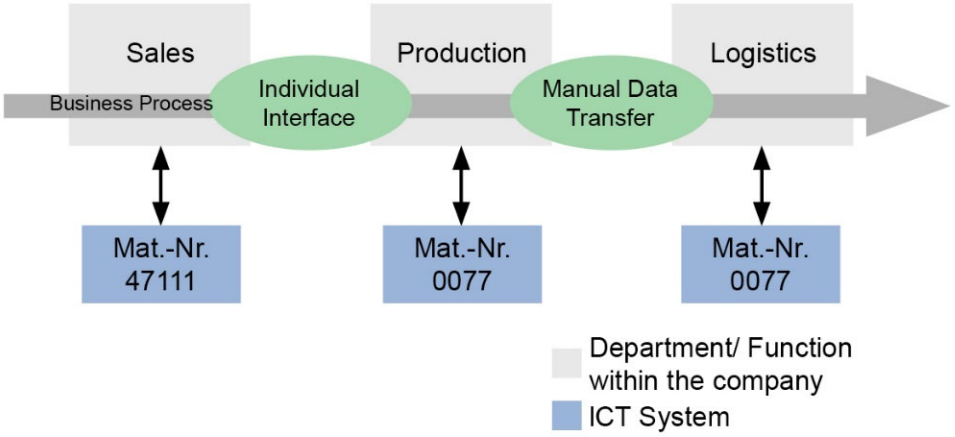


Fig. 5.10: Non-Integrated Application System.

5.4 ERP Systems

- Comprehensive **business application software packages** for resource planning in companies.

- **Key features:**
  - Standard software
  - Central database
  - Real-time availability
  - Modular design

Fig. 5.12: Typical Components of ERP Systems.

5.4 ERP Systems

SAP ERP: Sales Order Management

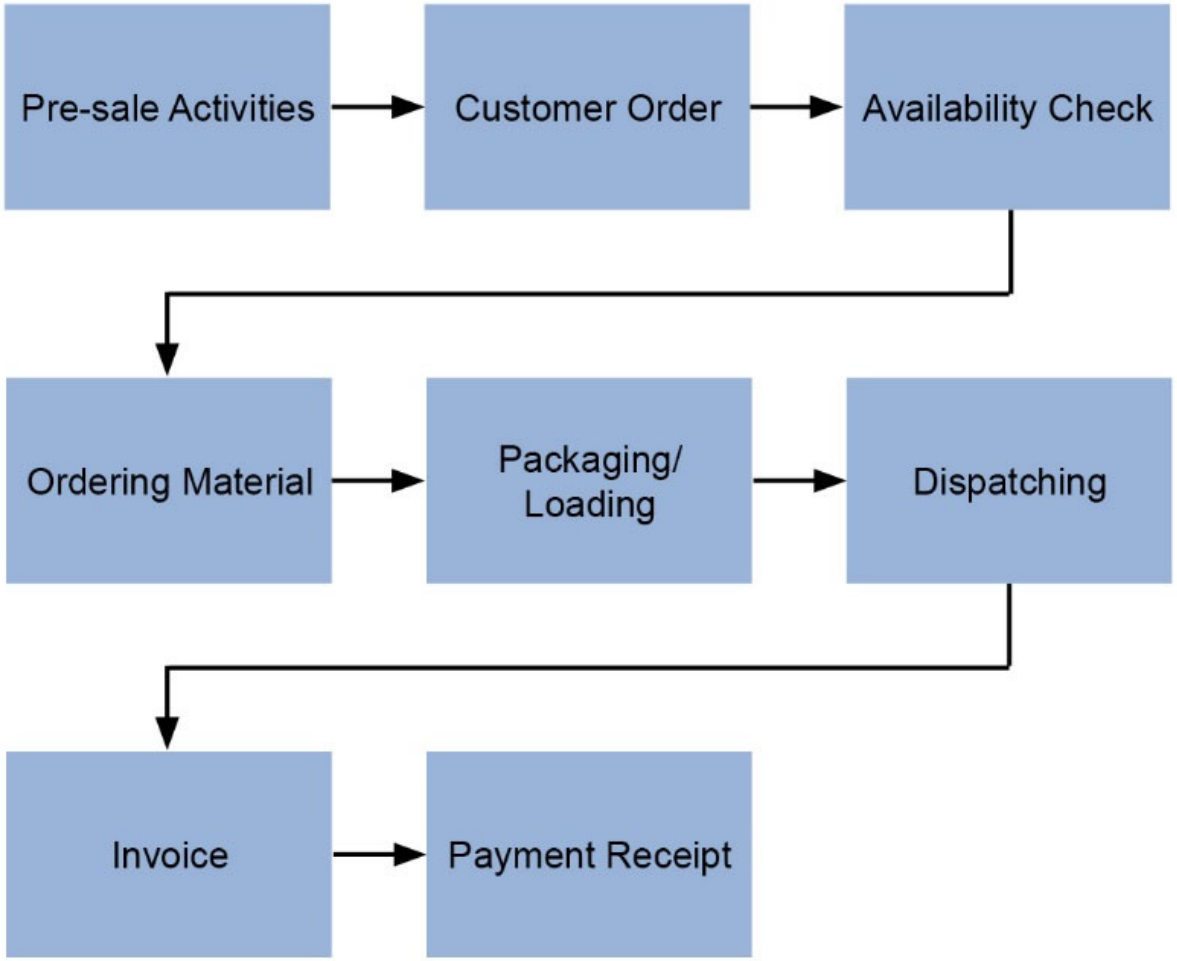


Fig. 5.16: Sales Order Management.

5.4 ERP Systems

SAP Business Suite

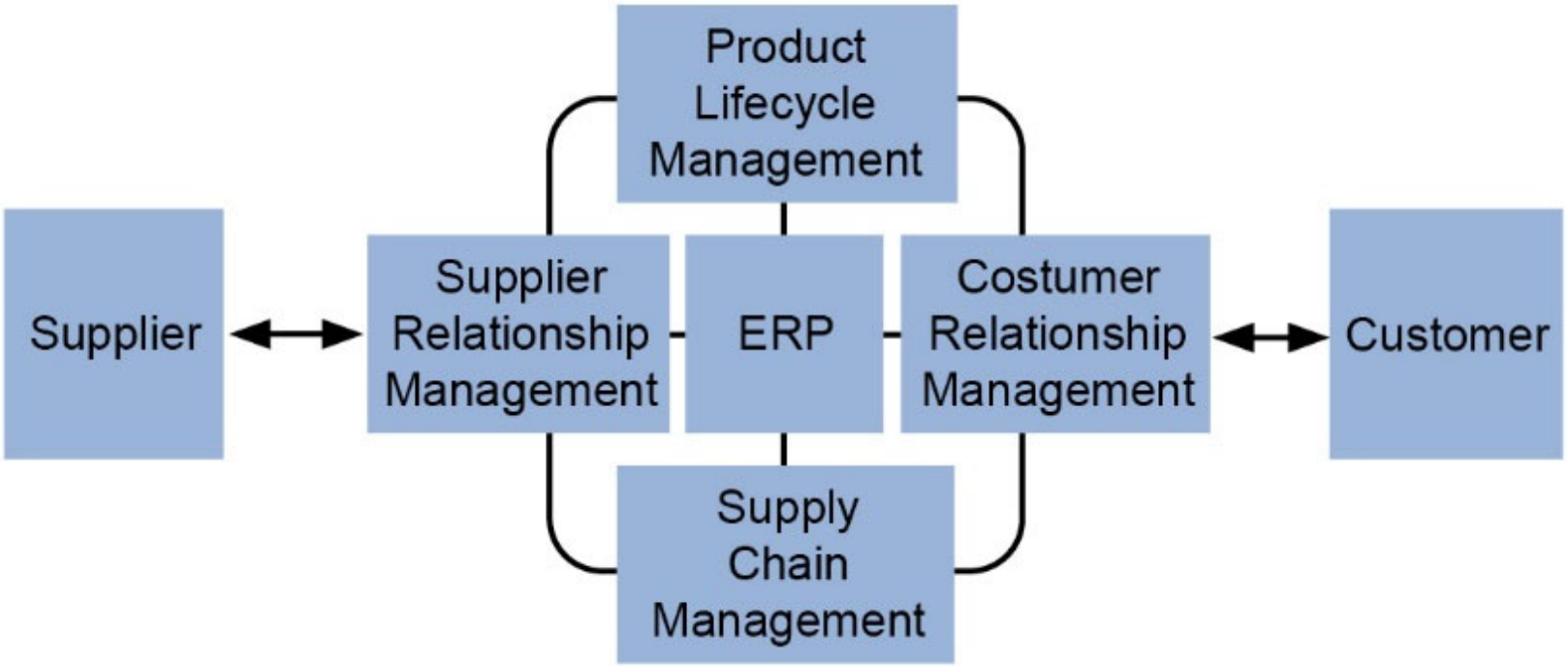


Fig. 5.13: ERP Systems as part of an Application Suite.

Contents

9.1 Subject & Objectives of

Software Engineering

- Design or creation of a software system
- Development, i.e., programming of a software system making use of programming languages or development tools
- Procurement of a software system from software companies and customization and integration of the software into the company’s software system

9.2 The Design Process

- Three phases in the design process of software (life cycle)

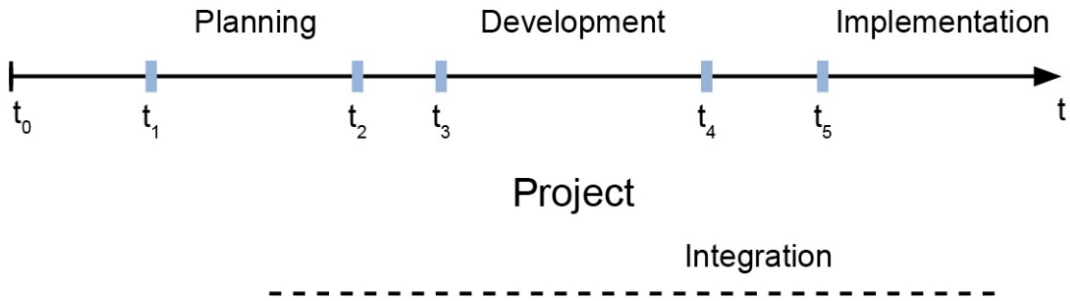


Fig. 9.1: Phases of a Design Process.

- t_0 = Emergence of idea
- t_1 + t_2 = Planning phase
- t_3 = Beginning of the development process (in case of a favorable decision)
- t_4 = Completion of the development process
- t_5 = Implementation of software, followed by continuous maintenance and control

9.2 The Design Process

Process Models

- Process models structure the software engineering process. Some of them are the spiral model, phase model, Extreme Programming and Scrum.

- **Spiral Model**
  1. Description of key requirements and objectives
  2. Evaluation of the listed solution options followed by the formulation of a strategy for risk avoidance/ minimization.
  3. Creation and evaluation of an interim software solution.
  4. Planning of the next iteration of the software engineering process.

9.2 The Design Process

Sequential Phase Model

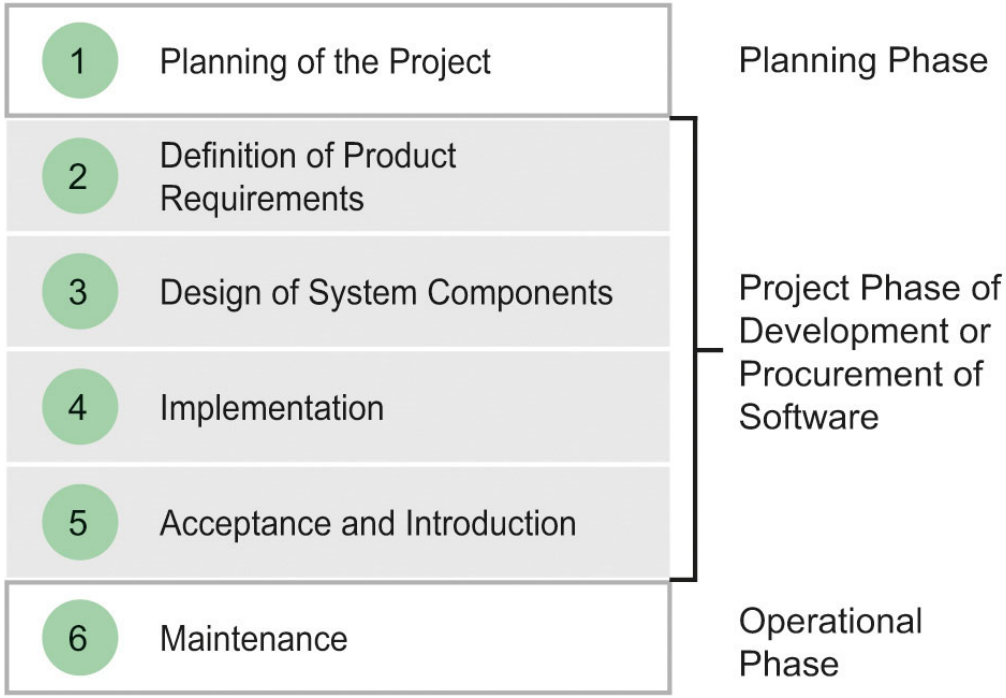


Figure 9.3: Phases in Software Engineering.

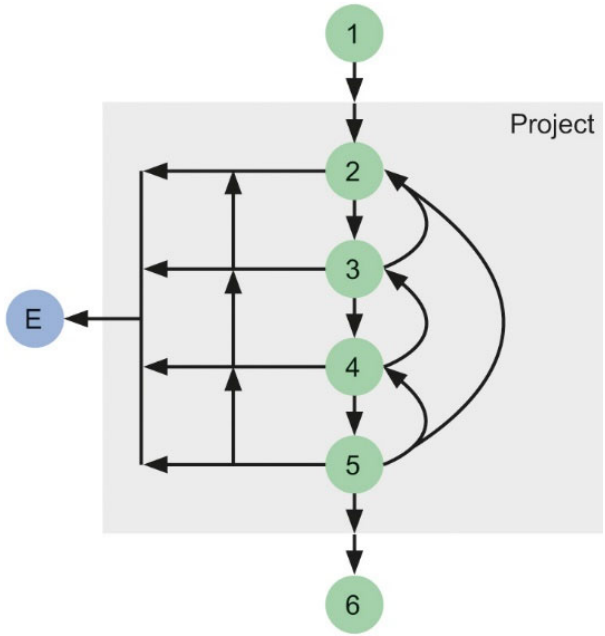


Figure 9.5: Variations of the Software Engineering Process.

9.2 The Design Process

Excursus: Specification Sheet / Requirement Specs.

- Tasks and discussion questions:
- What is the purpose of a specification sheet/ requirement specifications?
- Which components does a specification sheet/ requirement specifications have?
- What is the difference between the specification sheet and requirement specifications?
- Which benefits arise when preparing a specification sheet/ requirement specifications?

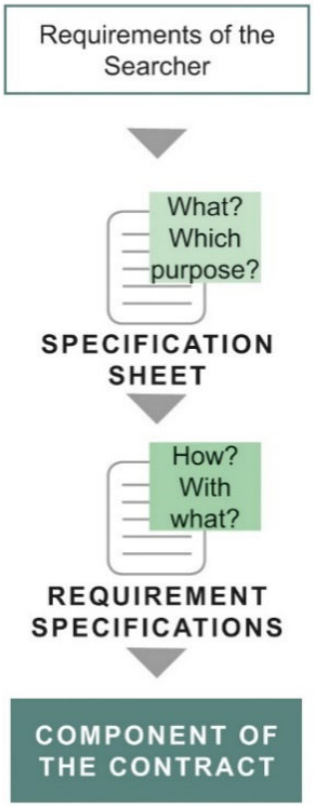


Figure 9.4 Specification Sheet and Requirements Specifications.

9.2 The Design Process

Agile Process Models

- Provide a more flexible project process (as opposed to classic approaches)
- Design and implementation of prototypes at early stages
- Prototype software versions are assessed and refined in regular tests

Fig. 9.6: The 12 principles of agile software development [Bec+01].

9.2 The Design Process

Agile Process Models

- **Example: Extreme Programming (XP)**
  - Flexible project course with periodic project revisions and a recursive structure
  - Key principles of XP: continuous communication, customer integration, simple functionality and teamwork

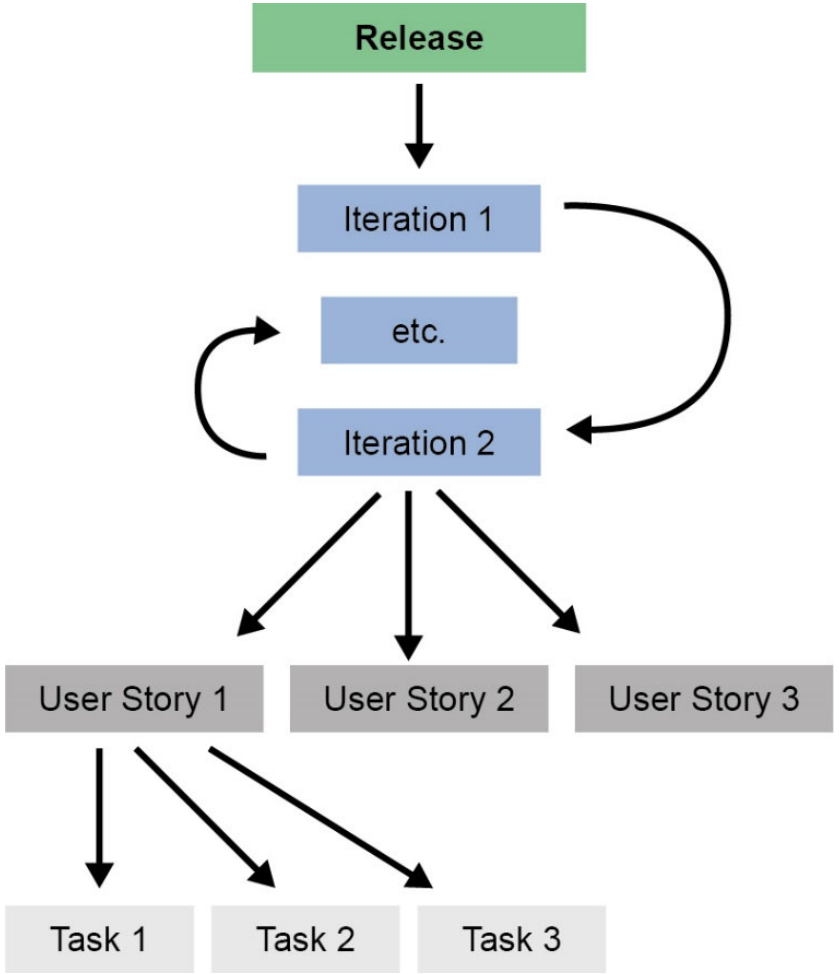


Fig. 9.7: XP: Release, iterations, user stories, and tasks (based on [Fern06]).

9.2 The Design Process

Scrum Framework

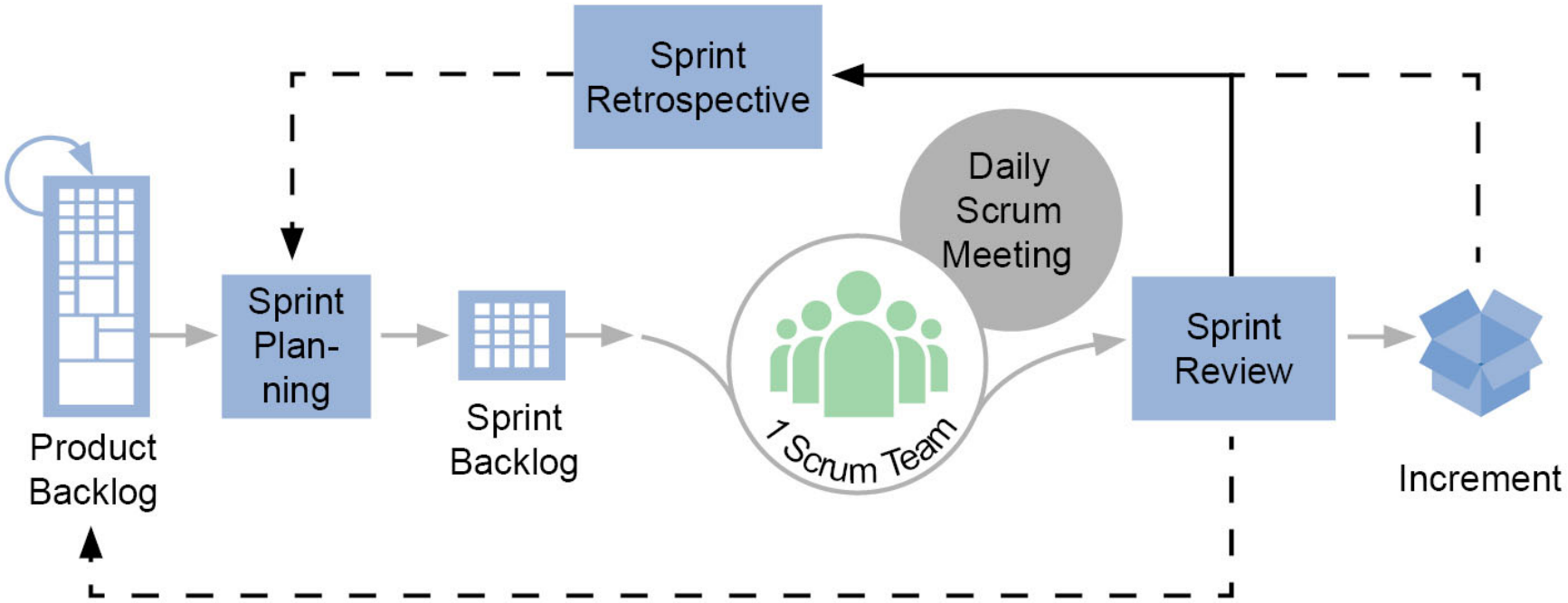


Fig. 9.8: Scrum Framework [Scru19].

9.2 The Design Process

Relevant Factors

- Which process model works best and how well a project runs depends on several different factors:
  - **The complexity of requirements**
    - Influenced by the amount of available data, existing structure and project scope
  - **The knowledge base**
    - Team members involved: IT specialists, software developers, planners, managers and users
  - **Information technology**
    - Programming languages and development software as well as hardware and operating systems
  - **Strategic (IT) objectives**

9.3 Objectives in Software Engineering

From Cost Efficiency to Usability

- **Cost Efficiency**
  - The key objective for management in a software engineering process
  - Maximize the benefits and minimize cost
- **Performance**
  - The key objective for IT experts
  - The system must fulfill the technical requirements and offer maximum performance with regard to robustness or processing speed
- **User Quality**
  - Important to the end users of the system
  - Quality of the software system must be recognized by users to make them work with it
- **Usability**
  - The software functionality should be transparent, easy to understand and comply with data protection laws

9.4 An Overview of Programming Languages

- Programming is carried out in a programming language
- Programming languages tend to be grouped in generations according to their chronological development:
  - 1st generation: machine languages
  - 2nd generation: assembler languages
  - 3rd generation: higher programming languages
  - 4th generation: 4GL language and alternative programming languages

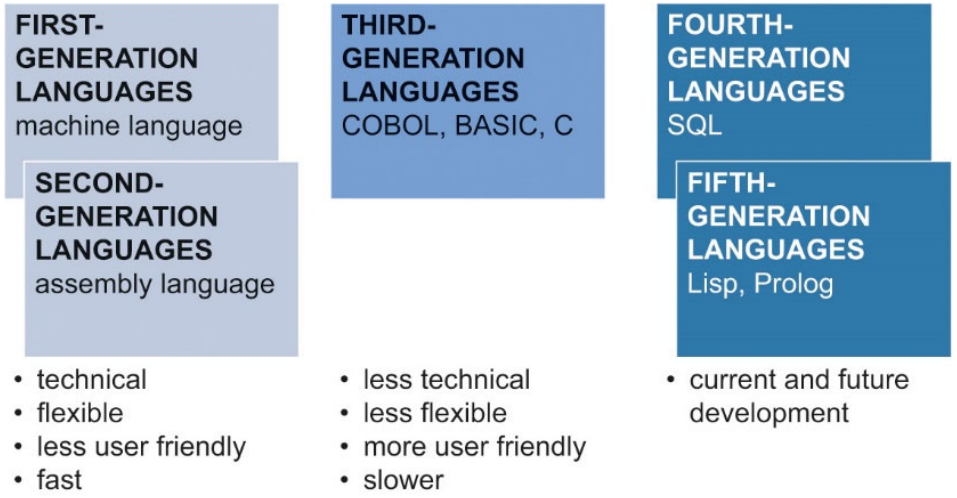


Figure 9.9: Generations of Programming Languages.

9.4 An Overview of Programming Languages

1st Generation / 2nd Generation

- Machine languages and assembler languages are **machine-based languages**
- The CPU can only process machine language
- **Machine languages**
  - All commands and operators are encoded in binary (in ones and zeros)
  - Each machine language is specifically designed for the processor
  - Hard to read and prone to errors
- **Assembler languages**
  - Allow the user to write in simple, easy to remember abbreviations …
  - … which are translated into machine code by an assembler
  - Also designed to work with a specific processor

9.4 An Overview of Programming Languages

Machine and Assembler Language

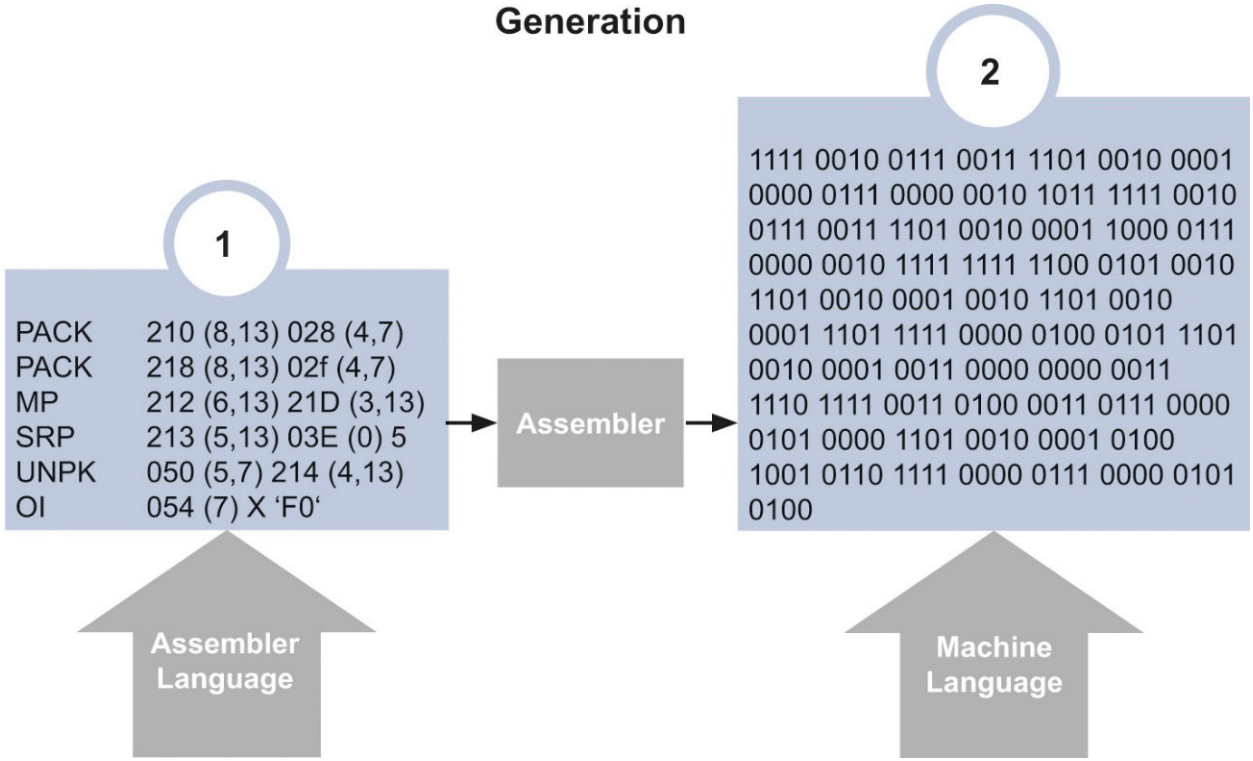


Fig. 9.10: Interaction of Assembler Language and Machine Language.

9.4 An Overview of Programming Languages

Higher Level Programming Languages

- Higher level programming languages allow more abstract commands and are **not dependent** on a specific processor (referred to as **problem-based languages**)
- The translation of higher programming languages into a machine language is done by a translator, e.g., a **compiler**

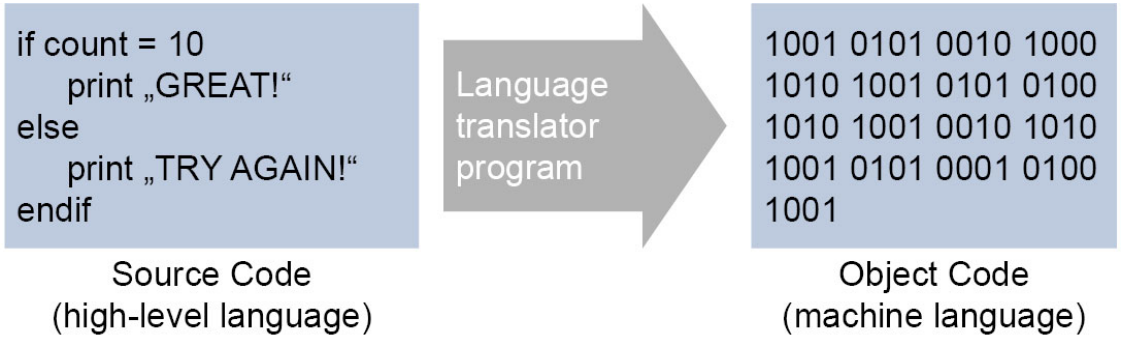


Fig. 9.11: Language Translator (Compiler).

9.4 An Overview of Programming Languages

4GL Languages

- **4GL languages**
  - Allow the programmer to write programs using very few commands and without requiring a deep knowledge of programming
  - One example is the database language SQL (Structured Query Language)
    - SQL compresses long program codes by using concise commands
      Example: SELECT statement (also see “Database Systems”)
- **Alternative programming languages**
  - Designed to work in specific contexts, such as artificial intelligence (AI) languages, visual languages, etc.

9.5 Object-oriented Software Engineering

- Based on the **object-oriented approach**
  - Writing programs more in the way that humans think
  - The focus of the object-oriented approach is the **object**. It has **attributes** describing its **state**.
  - The state of an object (its data) is hidden (**encapsulated**)
  - **Methods** can be used to change the state of the object or to exchange information with the object‘s environment
  - **Polymorphism** is the ability of an object to take on many forms
  - **Classes** are a system of defining several similar objects in one go. A class defines a number of attributes and methods that can be conferred on objects.

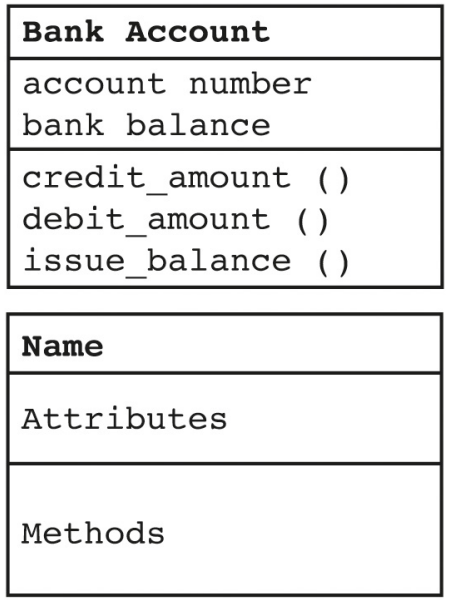


Fig. 9.13: A Class Diagram in UML Notation.

9.6 Expense Estimate

- The expense estimate is formulated in units of money, material consumption or working hours
- In the context of software engineering projects, the resource *employee* combined with the cost factor *time* represents the expense, measured in **employee months**
- **Expense estimates** seek to assess project expenses by early consideration
- The estimate accuracy is compromised by some basic problems and influence factors

9.6 Expense Estimate

Basic Problems

- **Accuracy of estimate:** Estimates are projections
- **Subject of the estimate**
  - Accuracy of estimate is influenced by the wealth of information (on the object)
  - Precise information is costly
  - Expense estimate takes place in the planning phase (normally, lack of information)
- **Documentation:** Poor documentation makes it almost impossible to harvest the experience of past projects
- **Legacy data:**
  - Lack of comparative data may cause miscalculations
- **Attitude of project participants**
  - Project participants often perceive expense estimates as irrelevant

9.6 Expense Estimate

Influencing Factors

- **Cost**
  - All factors influencing the financial expense of the project, e.g., number of project members
- **Quantity**
  - Factors such as project scope, size and complexity
- **Quality**
  - Quality features of the software
- **Time**
  - Project duration which is largely determined by the number of project members

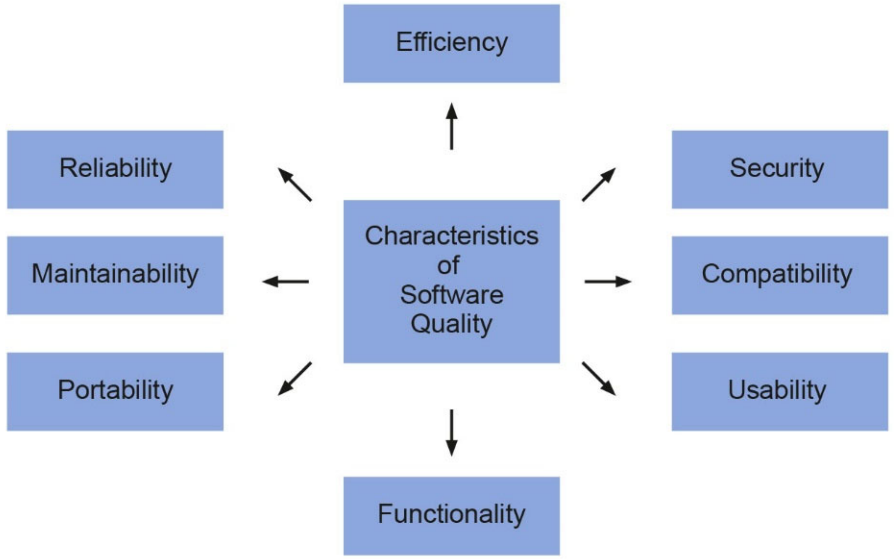


Fig. 9.16: Software Quality Features.

9.6 Expense Estimate

Sneed‘s Devil‘s Square

- The **four primary objectives** are **strongly interrelated** and in competition with each other
- **Productivity** is depicted as rectangle inside the square
- A positive change of one objective will have adverse effects on the others

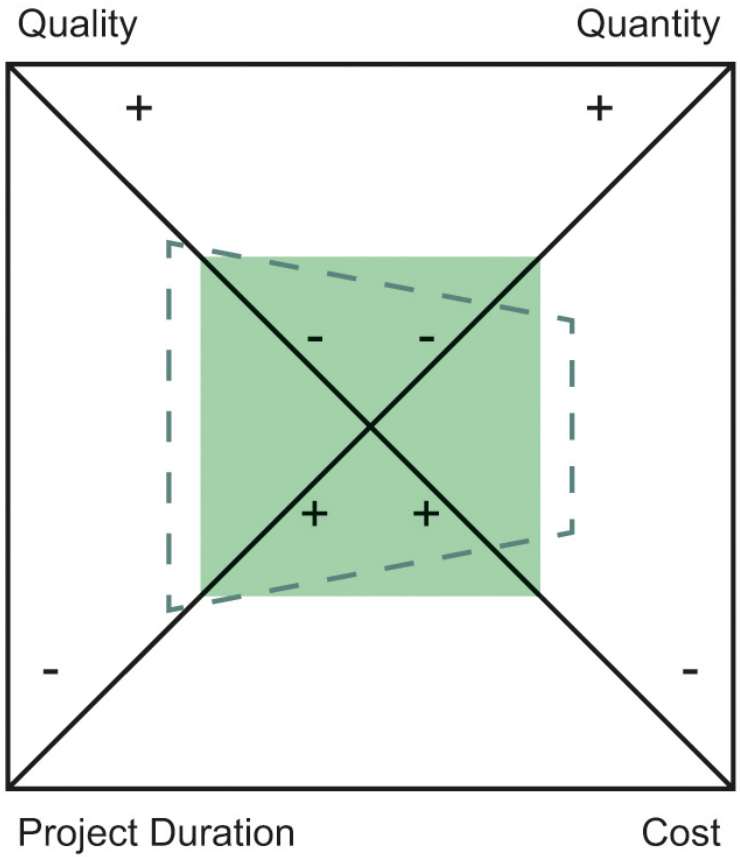


Fig. 9.17: Devil’s Square according to Sneed.

9.6 Expense Estimate

Methods and Procedures

- **Analogy method**
  - Calculates expense estimation based on previous projects (e.g., application area, product scope and programming language)
- **Relation method**
  - Based on a comparison with estimates of previous projects
  - The different factors are given weighted values
- **Multiplication method**
  - Subdivides the software system into components and categories
  - The expense of components is derived from previous projects and is weighted by means of the defined categories
- **Weighting Method**
  - Key influence factors must be determined and weighted using mathematical formulas
- **Percentage method**
  - Projecting values of the completed phase onto the next phase in the software engineering process
- **Parametric equation method**
  - Determines the correlation factor between influence factors and development effort
  - Gives an equation which indicates the influence factors with the highest correlation

9.6 Expense Estimate

Methods and Procedures

- **Function point model**
  - Assumption: the expense of the software development is strongly dependent on the complexity and scope of product realization
  - Therefore, the expense estimate is not based on system size but on the requirements specification (from the end user‘s point of view)

Components	Quantity	Classification	Weighting	Row Total
External Inputs		Simple	x3	=
		Medium	x4	=
		Complex	x6	=
External Inquiry		Simple	x3	=
		Medium	x4	=
		Complex	x6	=
External Outputs		Simple	x4	=
		Medium	x5	=
		Complex	x7	=
Internal Logical Files		Simple	x7	=
		Medium	x10	=
		Complex	x15	=
External Interface Files		Simple	x5	=
		Medium	x7	=
		Complex	x10	=
Total			E1	

Value Adjustment Factor (VAF) General System Characteristics (Influence Value from 0-5)	1. Data Communications	=
	2. Distributed Data Processing	=
	3. Performance	=
	4. Operational Configuration	=
	5. Transaction Rate	=
	6. On-Line Data Entry	=
	7. End-User Effiiciency	=
	8. On-Line Update	=
	9. Complex Processing	=
	10. Reusability	=
	11. Installation Ease	=
	12. Operational Ease	=
	13. Multiple Sites	=
	14. Facilitate Change	=
Total Sum of Influences	E2	=
Value Adjustment Equation VAF=0.65+[($\sum_{i=1}^{14}$ Ci)/100]	E3	=
Adjusted Function Points = E1*E2		=

Fig. 9.18: Example of a Tabular Calculation of the Function Point Model.